有效支持数据更新的 XML 索引研究

刘先锋[1],朱清华[1],陈凤英[2],丁继红[1]
LIU Xian-feng[1],ZHU Qing-hua[1],CHEN Feng-ying[2],DING Ji-hong[1]

1.湖南师范大学 数学与计算机科学学院,长沙 410081
2.湖南科技学院 教育科学系,湖南 永州 425100
1.College of Mathematics and Computer Science,Hunan Normal University,Changsha 410081,China
2.Department of Information Technology and Education,Hunan University of Science and Engineering,Yongzhou,Hunan 425100,China

LIU Xian-feng,ZHU Qing-hua,CHEN Feng-ying,et al.Research on XML index with effect supporting for data updating.Computer Engineering and Applications,2009,45(20):140-143.

Abstract: Efficient index is the key to improve efficiency of XML query.A large number of XML index algorithms have been proposed,however,most of them didn't support data updating.Motivated by this observation,CSSU Coding Scheme is improved,and a new index algorithm supporting data updating is proposed,which effectively supports single-path queries and branching queries.
Key words: Coding Scheme 1 of Supporting for Updating XML data (CSSU1);Extensible Markup Language (XML);XML data updating;XML query

摘 要: 高效的索引是提高 XML 数据查询效率的关键,目前已经提出了许多 XML 索引算法,它们大多数都不支持 XML 数据更新。对 CSSU 编码进行了改进,提出了一种新的有效支持数据更新的路径索引算法,该索引算法有效支持单支查询和多支查询。
关键词: 支持 XML 数据更新的编码方案 1;可扩展标记语言;XML 数据更新;XML 查询
DOI: 10.3778/j.issn.1002-8331.2009.20.042 **文章编号:** 1002-8331(2009)20-0140-04 **文献标识码:** A **中图分类号:** TP311

1 引言

随着 XML 的日益普及,越来越多的信息采用 XML 格式进行存储和交换。在 XML 模型中,已经提出了几种查询语言(如:Xpath、Xquery 等)用于处理树型结构。XML 索引是提高 XML 查询效率的关键,XML 数据的半结构化特点和 XML 查询的灵活性对现在的数据库索引方法提出了新的挑战。

希望把结构索引转换成能在不查询原 XML 文档的条件下适应单支查询和多支查询。之前的方法大致可以分为三类:(1)路径索引[1-4],为 XML 数据创建一个路径摘要。路径摘要可以提高单支查询速度,但在处理多分支查询时需要代价很大的连接操作。(2)节点索引:通过编码方案索引每一个数据节点[5-6]。节点索引方法对于任意一对节点的结构关系可以在常量时间内确定,但仅靠成对地比较是否满足查询有时效率较低。(3)基于序列的索引[7-8]:将 XML 文档和查询转换成序列,然后计算基于序列匹配的查询。在不需连接操作的前提下可支持灵活的查询,由于序列匹配不一定是树匹配,导致查询结果可能存在错误。

使用全局 ID 的先序索引方法,由于在值谓词和结构约束匹配间引用元素需要进行结构连接操作,这种引用没有任何语义,因为先序索引方法并不给出任何有关标签路径和标记名的信息。

针对以上问题,本文所作的工作如下:

(1)对 CSSU 编码[9]作了很大的改进,改进后编码的最大编码长度比原 CSSU 编码的最大编码长度成指数级缩短了,同时也提高了编码效率和查询效率,节省了大量的存储空间,能够更加有力地支持 XML 数据的更新。

(2)基于改进后的 CSSU 编码,提出了一种能有效支持更新的索引算法,并提出了新的基于路径索引的查询处理方法,可以加快查询速度,并减少早期处理过程中的搜索空间。节点编码保存了孩子双亲元素的详细信息,提高了访问元素双亲的速度,因此,这是一种处理 XML 结构约束的有效索引方法。

实验表明,所提出的编码方案能非常有效地支持索引的更新,提出的索引算法在大多数测试查询中性能较好。

2 XML 文档编码

为了有效支持 XML 查询,人们已经提出了很多编码方案,主要包括:位向量编码[10]、前缀编码[11]和区间编码[5-6]等。为了有效地支持 XML 数据更新,文献[9]提出了 CSSU 编码。

CSSU 编码方案改变了传统的采用数字编码的方法,CSSU

基金项目: 国家自然科学基金(the National Natural Science Foundation of China under Grant No.10571052);湖南省高校青年骨干教师资金(the Young Backbone Teachers Funds of Colleges and Universities in Hunan Province);湖南省教育厅科研资金(the Science Research Foundation of Education Department in Hunan Province)。
作者简介: 刘先锋(1964-),男,教授,研究方向:数据挖掘,电子商务技术;朱清华(1979-),男,硕士研究生,研究方向:数据挖掘,XML 数据库;陈凤英(1980-),女,讲师,研究方向:数据库;丁继红(1980-),女,硕士研究生,研究方向:数据挖掘和智能教学系统。
收稿日期: 2008-10-10 **修回日期:** 2008-12-22

编码是主要由字母组成,其主要思想:XML 树 D 中的每个节点赋予一个三元组($pcode$,$code$,$level$),其中 $level$ 表示节点在树中的层次,根节点的层数为 0,$pcode$ 表示双亲节点的编码,根节点的 $pcode$ 为空,只有根节点没有双亲,其他节点的 $pcode$ 为双亲节点的 $pcode$ 与双亲节点的 $code$ 相加(字符串相加)组成,$code$ 为节点在兄弟节点按字典序采用字母进行编码,根节点的 $code$ 为"a",每个节点的第一个孩子节点的 $code$ 为"b",第二个孩子节点的 $code$ 为"c",…,"z","zb","zc",…"zz","zzb","zzc"…。

CSSU 编码方案对 XML 数据更新的支持非常有效,不足之处就是当一个节点的孩子节点非常多时,会导致编码长度非常长,编码效率低,并影响查询速度。设某一节点的孩子节点的个数为 N,最大编码长度为 L,存在关系:$L=\lceil N/25 \rceil$。因为 1 个字节只能为 25 个节点编码,当一个节点有 100 000 个孩子节点时,编码的最大长度为 4 000 个字节。

2.1　改进的编码方案

对 CSSU 编码进行了改进,为了区别起见,将改进后的编码称为 CSSU1 编码,CSSU1 编码也是由一个三元组组成($pcode$,$code$,$level$),其中 $pcode$、$code$、$level$ 的含义与 CSSU 编码相同,不同的是 $code$="z"的下一个节点的 $code$ 为"bb",而不是"zb",$code$="zz"的右兄弟的编码为"bbb"。因此每个节点(包括根节点)的孩子节点的编码 $code$ 从左到右依次为"b","c",…"z","bb","bc",…"bz","cb","cc"…"zz","bbb","bbc",…"bbz","bcb"…"bcz","bdb"…"zzz","bbbb"…这样对于某节点的孩子节点数量很大时,CSSU1 编码最大编码长度比原 CSSU 编码最大编码长度缩短了指数级,设某一节点的孩子节点的个数为 N,最大编码长度为 L,则 $L=\log_{25}N$,4 个字节可以表示 390 625 个编码。对于 dblp 文档编码后如图 1 所示。

2.2　编码的更新

由于兄弟节点的 $pcode$ 和 $level$ 都是一样的,关键是 $code$ 的计算:如在节点 $N1$ 编码 $code$ 为"b"和节点 $N2$ 编码 $code$ 为"c"之间插入一个节点 N,由于 $N1.code+1$(指的是尾字符的 ASCII 码+1)后大于等于右兄弟 $N2.code$,则 $N.code$="b_b",如在"b_b"和"c"之间插入一个节点,则其 $code$ 为:"b_c",如果在"b_b"和"b_c"之间插入一个节点,则其 $code$ 为:"b_b_b"。这样在任意两个节点之间都可以插入无限个节点,而不存在编码会用完的问题。如果要在第一个节点 $N1$ 之前插入一个节点 $N0$,则 $N0.code$="a"+$N1.code$。

2.3　编码大小比较

CSSU1 编码大小的比较相对 CSSU 编码要复杂一些,而 CSSU 编码大小的比较相对简单,这是以编码长度作为代价的,有很多编码是从来都不会使用的,如编码中不会出现"bb","cb"等编码。

对于 CSSU1 编码大小的比较,设 n 为编码的最大长度,使用字符串截取函数 $right(str,n)$ 长度为 n 的字符串 s,其中 str=n 个"a"+节点编码 $code$,对于 $pcode$ 的比较采用相同的方法可得到。例如:有 3 个节点 $N1$、$N2$、$N3$ 的编码 $code$ 分别为"c"、"d"、"bb",假设编码长度最大为 4,$str1=right$("aaaa"+"c",4)、$str2=right$("aaaa"+"d",4)、$str3=right$("aaaa"+"bb",4),求得结果为 $str1$="aaac",$str2$="aaad",$str3$="aabb",很显然按照字母序,有 $str1<str2<str3$,若 $pcode$ 和 $level$ 相同,则 $N2$、$N3$ 是 $N1$ 右边的兄弟,$N3$ 是 $N2$ 右边的兄弟。

若 XML 文档经常更新,根据 2.2 部分节点编码中可能会存在一个或多个"_",当要进行这种类型的字符串的大小比较时,先将节点编码按"_"分裂为两个或更多的子字符串,对每个子字符串调用上面的函数,然后按顺序拼接起来,如在节点 $N1.code$="b" 和节点 $N2.code$="c" 之间插入一个节点 $N3$,则 $N3.code$="b_b",这 3 个节点按兄弟关系排列依次为:"b"、"b_b"、"c",假设编码最大长度还是为 4,按上面的方法求得的新字符串分别为"aaab","aaabaaab","aaac",很显然,兄弟次序从左到右为 $N1$、$N3$、$N2$,可以正确地表示兄弟之间的次序。如果不进行分裂则分别为"aaab"、"ab_b"、"aaac",这样字符串的大小为"aaab"<"aaac"<"ab_b",不能正确地表示兄弟之间的先后次序。

虽然 CSSU1 编码大小比较相对 CSSU 编码要复杂得多,但是因为其编码长度较短,所以比较速度比 CSSU 编码还是要快。

2.4　结构关系

父子关系:若 M 是 N 的双亲,则 $M.pcode+M.code=N.pcode$ 且 $M.level+1=N.level$;

祖先-后裔关系:若 M 是 N 的祖先,则 $M.pcode+M.code$ 是 $N.code$ 的前缀,且 $M.level<N.level$;

兄弟关系:若 N 是 M 的右兄弟,则根据 2.3 部分计算 $M.code$ 和 $N.code$ 求得新字符串分别为 $M.code'$和 $N.code'$,使得 $M.code'<N.code'$,且 $M.level=N.level$,$M.pcode=N.pcode$。

3　索引结构

对于 XML 数据树 D,生成路径摘要 S[12],S 上的每一个节点都称为一个组,每个组都确定地对应 D 上的一条标签路径。每个组包含了所有 D 上具有相同路径的节点,当组按它们的先序遍历序号有序时称之为有序路径摘要[13],图 1 的有序路径摘要如图 2 所示,一个方框表示一个组,框中用圆括号括起来表示具有相同路径的节点编码(没有加上 $level$),节点之间采用逗号隔开,方框旁边的数字表示组号,冒号后面的字符串表示组名,即方框中节点的元素名,从根节点顺着箭头方向往下走所经历过的组号组成一条路径,如组 3 的路径为:"/dblp/article/author"。

以往有关路径摘要的研究表明,基于路径概要的索引能快速地进行单支查询,但对于多支查询要付出很大的连接代价。因为他们对节点的编码都是采用数字的,路径摘要中没有记录

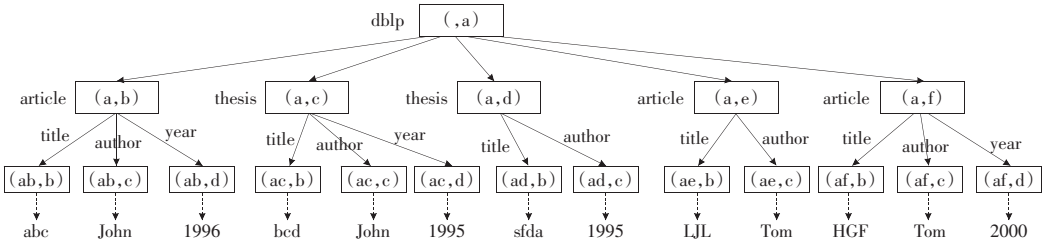

图 1　XML 文档 CSSU1 编码

节点之间的关系,如父子关系等。

根据本文提出的编码方案,从路径摘要中,还可以知道:父组中的哪个节点是子组中某个节点的双亲。如第 3 组中编码为(ae,c)的节点的双亲是第 1 组中编码为(a,e)的节点。因此本文提出的索引结构,不仅包含结构信息,还保留了孩子双亲元素的详细信息,有利于进行多支查询。

节点信息保存在数据库中,主要由两个表组成:*groups*(*gid*,*pgid*,*level*,*name*,*path*),*element*(*gid*,*pcode*,*code*,*value*)。节点编码等信息按组存储在数据库中,数据库中按组建立索引,可进一步提高查询效率。

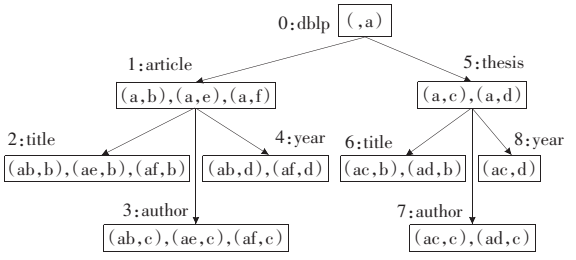

图 2 路径摘要

查询处理过程中,利用索引结构,避免了像传统方法那样逐个遍历所有文档元素进行查询判定,跳过了对无关元素的处理,从而提高了查询的效率。例如:在查询处理开始时,根据路径就可定位所有待选结果集所在组号;处理过程中,根据元素编码逻辑关系,可进一步减少对无关元素的处理,提高查询速度。

4 索引更新

4.1 插入 XML 文档

对 XML 文档进行插入操作时分为两种情况:

(1)待插入的文档片断 *newN* 作为某个 XML 文档节点 *N* 的第一个孩子节点插入,若 *N* 不存在孩子节点,则新节点编码 *newN.code*="b",若 *N* 存在孩子节点,设其第一个孩子节点 *N*1,如果 *N*1.*code* 的尾字符为"b",则 *newN.code*="a"+*N*1.*code*,否则设 *len*=*N*1.*code.length*,*newN.code*=*substring*(*N*1.*code*,0,*len*−1)+*N*1.*code* 尾字符的 ASCII 编码−1;

(2)待插入的文档片断 *newN* 作为某个 XML 文档节点 *N* 的第 *i* 个子节点插入,若 *newN* 作为 *N* 的最后一个孩子节点 *Nlast* 插入,则 *newN.code*=*nextcode*(*Nlast.code*),否则设 *str*=*nextcode*(*N.code*),若 *str*<*N.nextSibling.code*,则 *newN.code*=*str*,否则 *newN.code*=*N.code*+"_b"。

在一个节点之后插入一个节点的插入算法:

输入:待插入 XML 节点 *N*,待插入节点的双亲节点为 *pN* 的路径 *path*,待插入位置为 *pN* 的第 *i* 个子节点后插入。

输出:生成 *N* 的编码,并将其插入到相应的表中。

(1)根据 *pN.name* 和 *pN.path* 计算双亲节点 *pN* 的组号 *pN.gid*。

(2)根据双亲节点的组号 *pN.gid* 及其相关信息从 *element* 表中查找出双亲节点的编码(*pcode*,*code*,*level*)。

(3)在 groups 表中查找待插入节点 *N* 的组号,若不存在,则生成一个新的组,并插入到 groups 表中。

(4)在 element 表中找出节点 *N* 的所有子节点并按 2.3 部分的方法对 *code* 排列,根据插入位置相邻两节点的编码计算待插入节点编码,并插入到 element 表中。

(5)若 *newN* 有后裔节点,则可按先序遍历方法生成后裔节点的编码。

待插入的文档片断 *newN* 作为某个 XML 文档节点 *N* 的第一个孩子节点插入的算法与上面的算法类似,不同的地方在第 4 步,按对 XML 文档进行插入操作的第 1 种情况进行处理生成节点的编码,限于篇幅,不再累赘。

4.2 修改和删除 XML 文档

修改 XML 文档不影响其他节点的编码信息,因此只要在表中进行相应的更新操作即可。对于 XML 文档的删除,也不影响其他节点的编码信息,删除的编码可以留作将来插入节点时使用。

该索引算法的更新效率非常快,主要是因为插入一个节点或删除一个节点,不影响其他节点的编码,无需重新进行编码,从而大大提高了更新算法的效率。另外,在任意两个节点之间可插入的节点是没有限制的,从理论来说是可以插入无穷个节点,并且不影响其他节点的编码。Li-Moon 编码[5]通过预留序号来支持数据更新,但是预留序号的大小不好确定,而且预留序号总会有用完的时候,用完后不得不重新进行编码。而重新编码对于大文档或超大文档(如,几百兆,几 GB 大小的文档)来说,重新编码所需要的时间需要几个小时甚至几天时间,这是不能容忍的。

5 查询处理

M//*N* 的查询:*M* 和 *N* 可以是元素和简单路径,也可以是元素和简单路径的组合。首先在 groups 中找出 *name* 为 *N* 的最后一个元素名 *Nend* 的所有路径 *path*,并且 *path* 以 *M* 为前缀,以 *N* 为后缀的组号 *gid*,从数据库中找出所有组号为 *gid* 的记录,如 *M*="a/b/c",*N*="d/e/f",首先找出元素 f 的所有路径集 *paths*,然后对 *paths* 中的每一条路径 *path* 判断条件:*path* 是否包括 *M* 的字符,且 *N* 是 *path* 的后缀,如果条件成立,则存在这样的路径,根据其组号 *gid*,然后到 element 表中找出所有路径编号为 *gid* 的所有节点,否则不需要查找 element 表,减少了数据库的操作。

//*N* 型:只要从 groups 表中找出所有以 *N* 元素名为键值的组号 *gid*,然后到 element 表中查询所有组号为 *gid* 的节点。若 *N* 为简单路径,则从 groups 表中找出字段 *name* 为简单路径 *N* 的最后一个元素名的所有路径 *path*,并判断 *path* 是否以简单路径 *N* 结尾,如果是则根据其组号 *gid*,从 element 表中找出组号为 *gid* 的所有节点,继续判断下一路径,最后将所有满足要求的节点组合为结果。

M/*/*N* 的查询:将 * 去掉后与 *M*//*N* 型的查询类似,但又有区别,* 表示 *M* 元素的所有子节点,若 *M* 和 *N* 为简单路径,则简单路径 *M* 的最后一个元素的 *level* 和简单路径 *N* 第一个元素的 *level* 相差为 2。具体的查询方案为:首先从 groups 表中找出 *M* 和 *N* 最后一个元素的所有路径集 *path*1 和 *path*2,*N* 第一个元素 *N*1 的路径 *path*3,其中,*path*2 包含 *path*3 和 *path*1,*path*3 包含 *path*1,首先找出 *path*1 路径下的全部节点 *N*1;对 *path*3 中的每一条路径下的节点 *N*3 判断是否满足条件:(1)*N*3.*pcode* 包含 *N.pcode*+*N*1.*code*;(2)然后对 *path*1 中的每一条路径 *path* 判断 *N*3.*level*=*N*1.*level*+2;对 *path*2 中每条路径下的节点 *N* 满足如下条件:*N*2.*pcode* 的前缀为 *N*3.*pcode*+*N*3.*code*,将 *N*2 的节点进行合并即为结果。

M/*N* 这种类型的查询最为简单,只要找出 *M*/*N* 的组号

gid,然后按 *gid* 查询满足条件的节点即可。对于路径 *path*="/a/b/c/d/e"的绝对路径,只要根据路径 *path* 的组号 *gid* 就可以求得结果。

包含谓词的查询,从谓词处分开,对谓词左边和谓词右边的分别进行查询。然后进行连接。如:*path*="/dblp/article[author='John' and year>1994]/title"。

找到/dblp/article/author 的记录,使得 author 元素 author 的值为"John" and year>1994。然后再找 title,使 title 的 *pcode*=article 的 *pcode*+*code*,*level*=article 的 *level*+1;

复杂路径表达式一般都会带有一个或多个谓词条件表达式,这样路径表达式就会出现一个或多个分支,所以分解时以出现谓词的地方为分割点,将长路径分为多条小段的线性子路径,这些子路径的长度不受限制。因此,不管路径表达式有多长,只要将它分解为简单路径,就能正确地查找到所需的结果。只需要将该谓词条件中的属性与它的父节点做一次连接即可。对每条线性子路径查询完成后,要将它们的查询结果进行结构连接,从而得出这个复杂路径表达式的查询结果。

6　实验

为了计算 CSSU1 索引算法的效率,对 DBLP(http://dblp.uni-trier.de/xml/)数据集进行了测试,DBLP 文档的大小为 220 MB,总共包括 4 884 836 个元素。并且把 CSSU1 与目前性能较好的 Ctree[12]索引算法进行了比较。实验平台是 Windows XP 系统,Visual Studio 2008 C#,CPU 是 1.7 GHz,内存为 1 G。

CSSU1 编码:编码历时 53 min,编码最大长度为 10 个字符,平均长度为 6 个字符,组数为 129 个,element 表大小为 234 712 KB。

Ctree 编码:编码历时 64 min,组数为 129 个,element 表的大小为 264 776 KB。使用的查询用例如表 1 所示,其中 Q1 和 Q2 为单支查询,其余的 4 个为多支查询。CSSU1 和 Ctree 索引在 DBLP 上的查询性能比较,如图 3 所示。

表 1　查询用例

	查询路径	返回结果
Q1	/dblp/article	173 630
Q2	/dblp/book/author	1 484
Q3	/dblp/article[author="Frank Manola"]	22
Q4	//author[contains(./,"Frank Manola")	41
Q5	//article[./author="Frank Manola" and year>=1995	7
Q6	//article[./author="Frank Manola" and year=1995	2

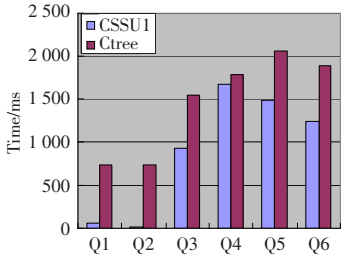

图 3　CSSU1 和 Ctree 索引在 DBLP 上的查询性能比较

7　结论

CSSU1 编码速度快,编码长度短,所提出的索引算法能有效地支持索引更新,在任意两个节点之间可以插入无穷个节点,并且不影响其他节点编码,按组建立索引,能跳过无关节点的处理,提高查询效率。

参考文献:

[1] Goldman R,Widom J.Dataguides:Enabling query formulation and optimization in semistructured databases[C]//Proceedings of the 23rd International Conference,VLDB,Aug 1997.San Francisco,CA USA:Morgan Kaufmann Publishers,1997:436-445.

[2] Cooper B,Sample N,Franklin M J,et al.A fast index for semistructured data[C]//Proceedings of the 27th International Conference on Very Large Data Bases,VLDB'01,September 2001.San Francisco,CA,USA:Morgan Kaufmann Publishers Inc,2001:341-350.

[3] Chung Chin-Wan,Min Jun-Ki,Shim K.APEX:An adaptive path index for XML data[C]//Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data,June,2002.New York,NY,USA:ACM,2002:121-132.

[4] Chen Qun,Lim A,Ong K W.D(k)-Index:An adaptive structural summary for graph-structured data[C]//Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data,June,2003.New York,NY,USA:ACM,2003:134-144.

[5] Li Quan-zhong,Moon B.Indexing and querying XML data for regular path expression[C]//Proceedings of the 27th International Conference on Very Large Data Bases VLDB'01,September 2001.San Francisco,CA,USA:Morgan Kaufmann Publishers Inc,2001:361-370.

[6] Zhang Chun,Naughton J,DeWitt D,et al.On supporting containment queries in relational database management systems[C]//Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data,2001:425-436.

[7] Wang Hai-xun,Park S,Fan Wei,et al.ViST:A dynamic index method for querying XML data by tree structures[C]//Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data,SIGMOD'03.June 2001.New York,NY,USA:ACM,2003:110-121.

[8] Rao P,Moon B.PRIX:Indexing and querying XML using Prunfer sequences[C]//Proceedings of the 20th International Conference on Data Engineering,ICDE'04,March 2004.Washington,DC,USA:IEEE Computer Society,2004:288.

[9] 刘先锋,朱清华,陈凤英.支持数据更新的 XML 编码方案研究[J].计算机工程与应用,2008,44(33):151-154.

[10] Wirth N.Type extensions[J].Transactions on Programming Languages and Systems,1988,10(2):204-214.

[11] Dietz P F.Maintaining order in a linked list[C]//Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing,STOC'82,May 1982.New York,NY,USA:ACM,1982:122-127.

[12] Zou Qing-hua,Liu Shao-rong,Chu W W.Ctree:A compact tree for indexing XML data[C]//Proceedings of the 6th Annual ACM International Workshop on Web Information and Data,WIDM'04,November 2004.New York,NY,USA:ACM,2004:39-46.

更　正

本刊 2009 年第 12 期第 236 页《集中供热系统中防止"窃热"行为的动态监控网络》一文的 DOI 应为"10.3778/j.issn.1002-8331.2009.12.075",文章编号应为"1002-8331(2009)12-0236-03"。